Light Speed Labelling : un nouvel algorithme d'étiquetage en composantes connexes.

Lionel Lacassagne[1;2], Maurice Milgram[1], Jean Devars[1]
[1]Laboratoire des Instruments et Systèmes
Université Pierre et Marie Curie BC 252
4 place Jussieu, 75252 Paris - FRANCE
lionel@lis.jussieu.fr devarsjmaum@ccr.jussieu.fr
[2]IMASYS
Le quadral 23 bis rue E. Nieuport
92150 Suresnes - FRANCE
lionel@imasys.fr

Résumé - Cet article présente un nouvel algorithme d'étiquetage en composantes connexe. son temps d'exécution est quasi indépendant des données pour une taille d'image donnée. De plus sa structure plus adaptée aux architectures complexes des processeurs et des DSP d'aujourd'hui, rend la version 8-connexe aussi rapide que la version 4-connexe. Enfin cet algorithme est optimum en ce qui concerne le nombre d'étiquettes créées et il peut être facilement et efficacement parallélisé.

Mots clés - adéquation algorithme - architecture, nouvel étiquetage connexe, RISC, DSP, temps réel.

I. Introduction

L'étiquetage en composantes connexes est la base de presque toutes chaînes algorithmiques en traitement d'image dès qu' il s'agit d'analyser des régions dans une scène. Plus généralement il est présent lorsqu'il faut regrouper des pixels connexes. On le retrouve donc aussi pour étiqueter des segments[3], en détection de contours, ou pour trouver les caractères en OCR. Si Rosenfeld [4] a révolutionné le domaine en proposant un algorithme ne comprenant que 2 passes sur l'image, nous proposons un algorithme en 3 passes, qui compte tenu de l'architecture des processeurs RISC actuels[1] ainsi que des DSP, est plus rapide que la version \classique" en 2 passes.

II. Etiquetage classique, étiquetage basé sur le codage RLC

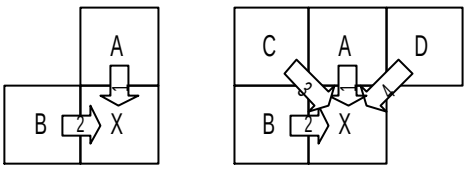

Figure 1 { étiquetage 4-connexe et 8-connexe

Les algorithmes standards utilisent soit l'approche pixel (Fig 1: comparaison de l'étiquette courante avec les étiquettes du voisinages)s) soit une approche basé sur le codage RLC (Run Length Coding, recherche de segments adjacents pour déterminer la connectivité des segments). Ces deux approches utilisent un grand nombre de tests, ce qui se traduit au niveau du processeur par un blocage du pipeline (et donc d'une perte de temps) car, bien sûr, le résultat d'un test est imprédictible. Enfin et ce n'est pas le moindre de ses défaut, l'approche pixel produit beaucoup d'étiquettes inutiles.

III. L'architecture des processeurs actuels

Les processeurs actuels RISC et DSP intègrent des mécanismes de plus en plus complexes pour accélérer l'exécution du code et le traitement des données : architectures superscalaires, su-

perpipelines, VLIW, prédiction de branchement, exécution dans le désordre, mémoires caches [5].

Les algorithmes classiques d'étiquetage ne sont plus du tout adaptés à l'architecture des processeurs. Leur structure les empêche d'être optimisés statiquement par les compilateurs et dynamiquement par les processeurs, et ce pour deux raisons principales :

- adressage dispersé des pixels menant à des défauts de lecture de la mémoire cache.
- présence de tests empêchant le pipeline logiciel et la vectorisation des calculs. Une prédiction de branchement erronée est d'autant plus pénalisante que le pipeline comporte d'étages, l'erreur étant détectée plus tardivement.

Une optimisation architecturale étant impossible, nous avons optimisé l'algorithme lui même, en tenant compte et en tirant parti de l'architecture des processeurs (instructions pipelinables/vectorisables, accès régulier aux caches).

IV. L'algorithme LSL

L'idée principale de l'algorithme LSL (Light Speed Labelling) est de remplacer tous ces tests (comparaison d'étiquette, tri par fusion pour les intersections de segments) par un ensemble d'instructions plus simples et surtout sans test.

La vitesse du LSL est basée sur les 3 points suivants :

- utilisation intensive du codage RLC
- introduction d'un étiquetage \relatif" et \absolu" permettant un accès direct (LUT) à l'étiquette associée aux segments intersectés
- une implémentation cyclique de la relation d'équivalence entre étiquettes, plutôt que l'habituelle représentation à \plat", par tableau.

Voici les différentes étapes de l'algorithme. Les explications seront illustrées par l'exemple suivant (Fig 2).

1. Calcul des tables ER_i et RLC_i

La table ER_i contient, pour chaque ligne L_i, l'étiquette relative e_r de chaque segment (objet avec un numéro impair) et de chaque non-segment, c'est à dire le fond (avec un numéro pair). Ce calcul peut être fait rapidement et sans aucun test, car l'étiquette e_r est en fait un compteur de transition (front montant, front descendant) et que ces transitions sont facilement détectées par un ou exclusif (XOR).

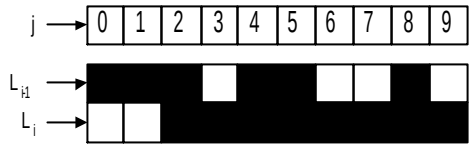

Figure 2 { Lignes L_i, L_{i-1}

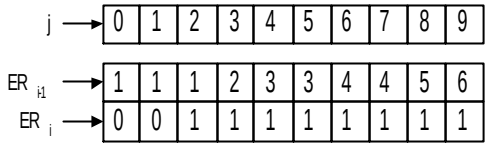

Figure 3 { table ER

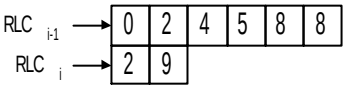

Figure 4 { table RLC

La table RLC_i contient les index de début et de fin de chaque segment [a::b]. Le $j^{ème}$ segment d'étiquette relative $e_r = 2j + 1$ aura comme bornes $a = RLC_i[2j]$, $b = RLC_i[2j + 1]$. Le segment courant aura une intersection avec les segments numérotés $e_{r0} = ER_{i-1}[a]$ $e_{r1} = ER_{i-1}[b]$. Afin de ne pas pointer vers le fond, les étiquettes e_{r0} / e_{r1} seront éventuellement ajustées pour pointer vers le précédent segment, ou le suivant.

Pour réaliser un étiquetage 8-connexe, ce qui est équivalent à tester les étiquettes en diagonales (NO, NE), il suffit d'incrémenter a de 1 et de décrémenter b de 1, juste avant de regarder dans la LUT ER_{i-1}. Cela rend la version 8-connexe aussi rapide que la version 4-connexe.

2. Calcul de la table of ERA_i

La table ERA_i associe les étiquettes relatives

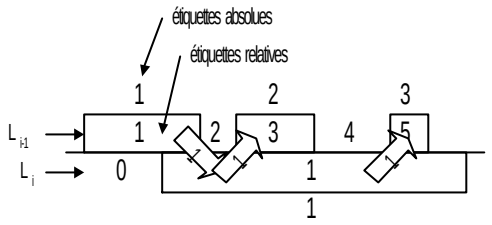

Figure 5 { étiquettes relatives et absolues

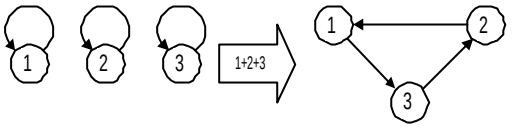

Figure 6 { cycle d'équivalence

aux étiquettes absolues de chaque ligne. La table est calculée durant la propagation vers le bas des étiquettes. L'étiquettes relative est associée à l'étiquette absolue du premier segment intersecté :

$$e_a = 1$$
$$\begin{cases} a = RLC_i[0] = 2 \;)\; a = 1 \\ b = RLC_i[1] = 9 \;)\; b = 9 \end{cases} \quad \begin{cases} e_{r0} = 1 \\ e_{r1} = 6 \;)\; e_{r1} = 5 \end{cases}$$
$$e_a = ERA_{i-1}(e_{r0}) = 1 \,,\; ERA_i(e_r) = e_a$$

3. Calcul de la table cyclique EQ

Résoudre les problèmes d'équivalence est complexe (ajouter un item à une classe, ou plus généralement, fusionner deux classes d'équivalence, voir calculer la fermeture transitive d'un graphe qui est un problème NP-complet). L'implémentation habituelle des classes d'équivalence sous forme de tableau, donc \à plat" a été remplacée par une implémentation sous forme de graphe cyclique.

Durant la fusion, e_a, doit être associé avec les étiquettes absolues e_{ak}, elles même associées aux étiquettes relatives :

$e_{rk} \in [e_{r0} + 2 :: e_{r1}]$, $e_{ak} = ERA_{i-1}[e_{rk}]$

$fusion(e_a; e_{ak})$

En reprenant l'exemple de la section précédente :

$e_{rk} \in [e_{r0} + 2 :: e_{r1}]$

$e_{rk} \in \{3; 5\}$, $e_{ak} \in \{2; 3\}$

La résolution des équivalence entre ces trois étiquettes sera traitée comme schématisé par la figure 6.

4. Calcul de la ligne EA_i

Les étiquettes absolues de la ligne L_i sont simplement obtenues par la lecture de la LUT ERA_i: :

$EA_i = ERA_i(ER_i)$

5. Etiquetage final

Lorsque l'image a été entièrement balayée, la résolution complète des classes d'équivalences a lieu, les étiquettes absolues sont renumérotées (afin d'avoir une numérotation continue). EQ est donc \réduite" pour donner A. L'image est alors entièrement ré-étiquetée :

$EA_i = A(EA_i)$

A. Analyse des performances de l'algorithme LSL

Nous avons comparé notre algorithme à celui utilisant l'approche pixel. La version \classique" de l'algorithme pixel utilisant une représentation à plat des classes d'équivalences, nous l'avons modifié en lui adjoignant notre représentation \cyclique". Ces algorithmes utilisent des étiquettes codées sur 16 bits car l'approche pixel dépasse très souvent la limitations à 256 étiquettes de la version 8 bits. Ces algorithmes ont été implémentés en 4 et 8-connexe. Nous avons décider de réaliser les tests sur quatre types d'images : petites/grandes régions, peu/beaucoup de caractères. Nous donnons les résultats mesurés en millisecondes sur un Pentium II 400 MHz, pour des images de taille 512 × 512.

boulons

logo

casse 72

casse 28

	boulons		logo	
	4C	8C	4C	8C
pixel	32	64	26	50
pixel+cycle	20	24	68	42
LSL 16bit	21	21	20	20
LSL 8bit	12	12	11	11
gain LSL/pixel	2.7	5.3	2.4	3.7

	casse 72		casse 28	
	4C	8C	4C	8C
pixel	20	30	20	30
pixel+cycle	24	20	19	23
LSL 16bit	20	20	23	23
LSL 8bit	12	12	-	-
gain LSL/pixel	1.7	2.5	0.9	1.4

Notre algorithme est de 2.4 à 5.3 fois plus rapide pour les images de type \blob". Même pour les images d' OCR, LSL est plus rapide.

Ces résultats montrent qu'un algorithme faisant trois passes sur l'image peut être plus rapide que les 2 passes habituelles depuis 1960, et que plus que tout, ce n'est pas le nombre d'accès pixel qui compte, mais la manière de les faire et l'alimentation du processeur.

V. Une seconde application : le seuillage par hystérésis

Si l'utilisation première de cet algorithme est l'étiquetage de région, il peut aussi être utilisé pour réaliser un seuillage par hystérésis. Pour cela, il suffit, après un seuillage bi-niveau (seuil bas - seuil haut) de déterminer le niveau (bas ou haut) de chaque segment. Soit l'exemple de la figure 7:

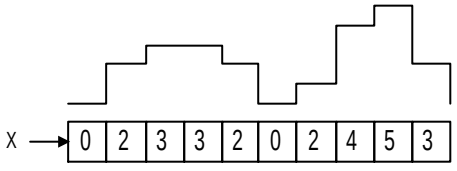

Figure 7 { profil d'une ligne en niveaux de gris

En seuillant aux niveaux 2 et 4, avec comme codes de niveau 1 et 3, nous obtenons (fig 8) :

Les codes 1 et 3 ont été choisis afin de ne détecter que les fronts entre le niveau zéro et un autre niveau (et pas les fronts entre niveau bas et niveau haut). La transition t est obtenu par $t = (x_i \text{ XOR } x_{i-1}) \text{ AND } 1$. Le niveau du segment est calculé, sans faire de test, avec un ou inclusif (OR) entre le niveau de tous les pixels du segment.

La propagation de ces niveaux ne se faisant qu'à la fin, lors de la résolution des équivalences, il n'influe guère sur les temps de calculs.

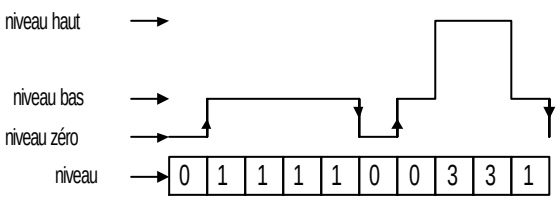

Figure 8 { seuillage bi-niveaux

VI. Conclusion

Dans cet article, nous avons proposé un algorithme très rapide d'étiquetage en composantes connexes. Cet algorithme en utilisant un étiquetage relatif à chaque ligne basé sur le codage RLC, et par ailleurs, une représentation cyclique des classes d'équivalences permet d'obtenir un fonctionnement plus régulier du processeur amenant un gain de performances important. Cet algorithme, dont le temps de traitement sur Pentium II 400 est largement inférieur aux 40 ms de la cadence vidéo, nous permet de compléter notre chaine de détection de contours en temps réel sur RISC et DSP[2] pour des images 512 × 512.

References

[1] J.L. Hennessy D.A. Patterson \Architecture des ordinateurs" traduit par D. Etiemble, Thomson Publishing

[2] L. Lacassagne, F. Lohier, P. Garda \Real time execution of optimal edge detectors on RISC and DSP processors", ICASSP '98

[3] J.F. Quesne, D. Demigny, J. Devars J.P. Cocquerez \Architecture temps réel pour la détection des contours et l'étiquetage des régions", 8ème Congrès RFIA, pp65-80, Lyon (1991)

[4] A. Rosenfeld, J.L. Platz \Sequential operator in digital picture processing", Journal of ACM vol 13 n°4 471-494 (1966)

[5] A. Seznec, T. Lafage \Evolution des gammes de processeurs MIPS, DEC Alpha, Power PC, SPARC, x86 et PA-RISC", Publication interne Irisa, n°1110, (1997)